ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

APLICACIÓN GPS DE REALIDAD AUMENTADA PARA EL CAMINO DE SANTIAGO EN DISPOSITIVOS ANDROID

Ricardo Meana de la Llave

Jesús Villadangos Alonso

Pamplona, Septiembre de 2012

ÍNDICE

ÍNDICE DE FIGURAS

1 – INTRODUCCIÓN

En los tiempos actuales, están apareciendo continuamente nuevas plataformas de programación muy diversas. Entre ellas, se encuentran en continua progresión las plataformas de dispositivos móviles, situándose a la cabeza iOS y Android.

La plataforma iOS de Apple es la más utilizada en el mundo debido a la multitudinaria clientela que tienen con sus iPod, iPhone y iPad, pero el Android de Google le sigue muy de cerca gracias a la amplia gama de dispositivos que emplean su sistema operativo.

Actualmente se están desarrollando para los dispositivos móviles aplicaciones que emplean tecnología conocida como realidad aumentada. Dicha tecnología hace uso de la vista del móvil para mostrar diferentes contenidos, es decir, combina una imagen real con contenido digital.

La realidad aumentada es una tecnología que a pesar de existir algunos juegos realizados con ella, la mayoría de las aplicaciones que la utilizan lo hacen para mostrar contenido según la geolocalización del usuario y la dirección a la que se encuentra mirando. Esto hace que sea muy interactiva, ya que el usuario debe moverse para aprovechar toda la experiencia que ésta ofrece.

El objeto de este proyecto es unir las partes comentadas anteriormente, es decir, una aplicación para plataformas móviles (en este caso para Android), tecnología de realidad aumentada y la mayor interactividad posible con el usuario. La conclusión a la que se ha llegado es a la realización de una aplicación que sirva de guía para realizar el Camino de Santiago mostrando al usuario diferente información en realidad aumentada según lo va realizando.

2 – ARQUITECTURA DEL SISTEMA

En este apartado se comentará la arquitectura del sistema operativo Android y de cómo funcionan las aplicaciones que instalamos en un dispositivo con este sistema operativo. La siguiente imagen muestra la arquitectura que sigue el sistema.

Ilustración 1 - Arquitectura de Android

Como se puede ver la arquitectura del sistema de Android **[1]** sigue la estructura de una pila, y esto es debido a que elementos de una capa emplean elementos de capas inferiores para realizar sus funciones. A continuación se explican las capas con más detalle:

Kernel de Linux: es el núcleo del sistema operativo, actúa como abstracción entre el hardware y el resto de capas, y se encarga de gestionar los recursos del teléfono y del sistema operativo.

Librerías: son las bibliotecas nativas de Android, desarrolladas específicamente para el hardware del dispositivo y proporcionan funcionalidad para tareas que se ejecutan frecuentemente. Dentro de estas librerías nos encontramos con algunas que intervienen en la aplicación: OpenGL (motor gráfico con el que se muestra la dirección a seguir en el Camino) y SQLite (bases de datos donde almacenamos las diferentes rutas y los puntos de interés a mostrar).

Entorno de ejecución: está compuesto por otras librerías y su componente principal es la máquina virtual Dalvik, encargada de ejecutar las aplicaciones de nuestro dispositivo.

Framework: clases y servicios que las aplicaciones emplean directamente para realizar sus funciones. La mayoría de sus componentes son librerías escritas en Java que utilizan recursos de capas anteriores gracias al entorno de ejecución.

Aplicaciones: en esta capa es donde se encuentran todas las aplicaciones del teléfono.

Lo comentado anteriormente es el funcionamiento del sistema operativo de Android. A continuación se explicará como funcionan las aplicaciones de Android en cuanto a actividades y vistas. A diferencia del lenguaje de programación Java, que en un proyecto sólo podemos tener una clase que contenga un main, el código que se tiene que ejecutar al lanzar el proyecto, en un proyecto de Android podemos tener tantos "main" como queramos. En este caso, se llaman actividades.

¿Qué es una actividad? Una actividad es algo único que el usuario puede hacer. La mayoría de las actividades interactúan con el usuario, por lo que dicha clase es la encargada de crear una ventana en la que el usuario puede poner su interfaz y mostrar la información que desee. El sistema de Android maneja las actividades a través de una pila. Cuando una actividad nueva empieza, se coloca en la cima de la pila y pasa a ser la actividad en ejecución. La actividad que se encontrara ejecutándose anteriormente sigue haciéndolo, pero no se verá hasta que salgamos de la nueva actividad.

A continuación se muestra un diagrama que explica los diferentes estados por los que puede pasar una actividad. [2] Los cuadrados blancos son las funciones que podemos sobrescribir para realizar unas acciones u otras según lo que deseemos.

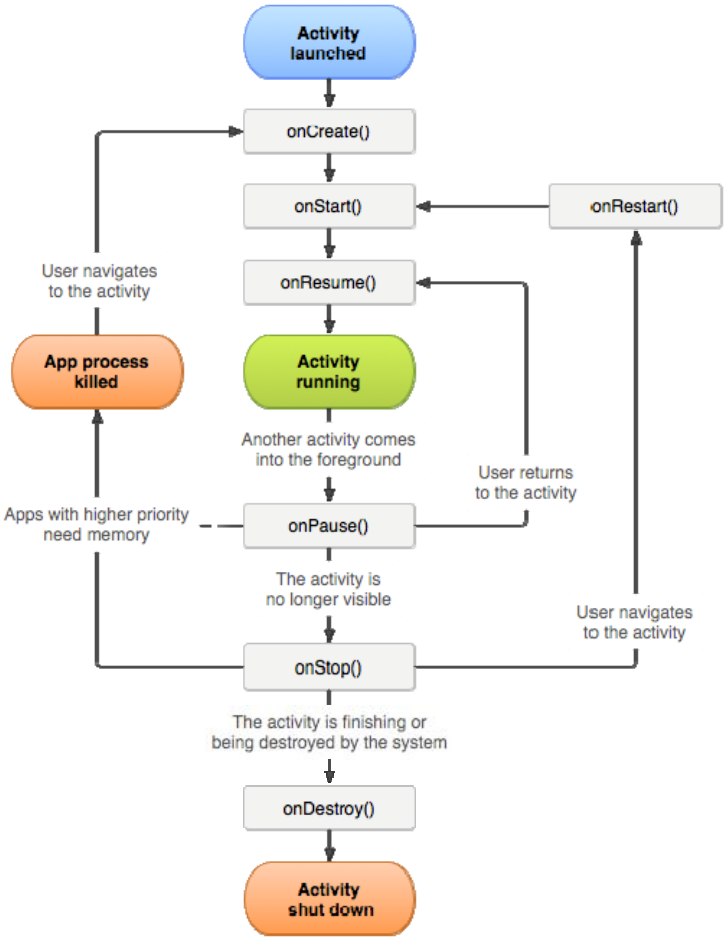

Ilustración 2 - Ciclo de vida de una actividad

Antes se ha comentado que una actividad puede crear una ventana para mostrar la interfaz de usuario. Dichas interfaces de usuario en Android se llaman vistas. Las vistas son bloques básicos donde colocar distintos componentes de las interfaces de usuario.

Una vista ocupa un área rectangular de la pantalla y es responsable de dibujar dichos componentes y de responder a los eventos.

En consecuencia y por lo comentado anteriormente, la aplicación se encuentra en la capa de aplicaciones, mediante el Framework y las librerías accede a los diferentes recursos que necesitamos: la cámara, que emplearemos para mostrar el contenido digital sobre lo que realmente ve el usuario, la tecnología OpenGL ES gracias a la cual mostramos el camino que debe seguir, la tecnología SQLite con la cual administramos las bases de datos de rutas y puntos de interés que empleará nuestra aplicación.

La aplicación comienza con una actividad que muestra en pantalla una imagen de presentación.

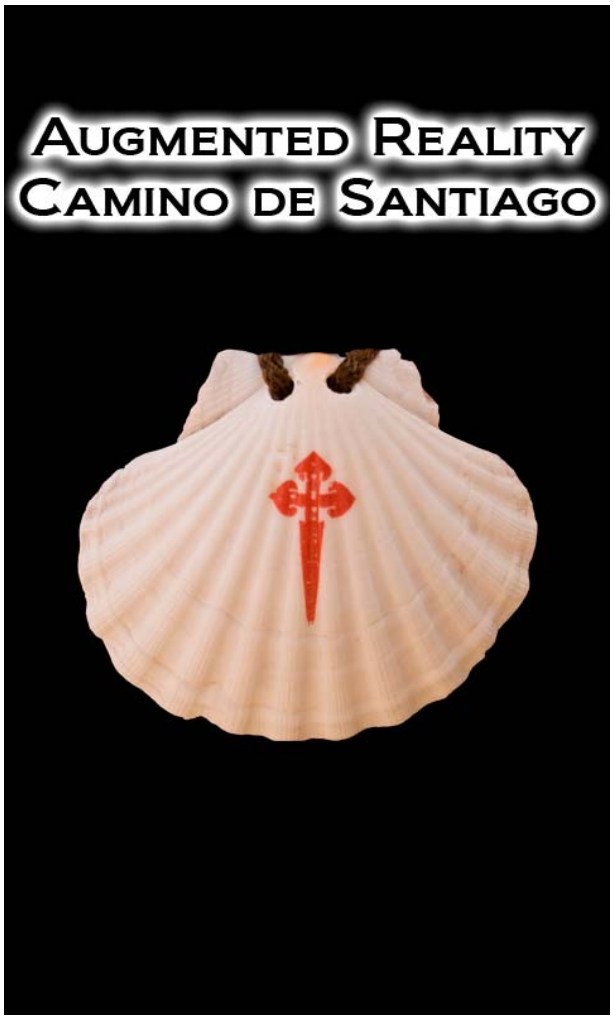

Ilustración 3 - Pantalla de inicio de la aplicación

Si pulsamos esta pantalla, se lanzará una nueva actividad que realizará una acción u otra en función de si es la primera vez que hemos ejecutado la aplicación. Esta actividad será una que nos permita configurar la aplicación en caso de ser la primera.

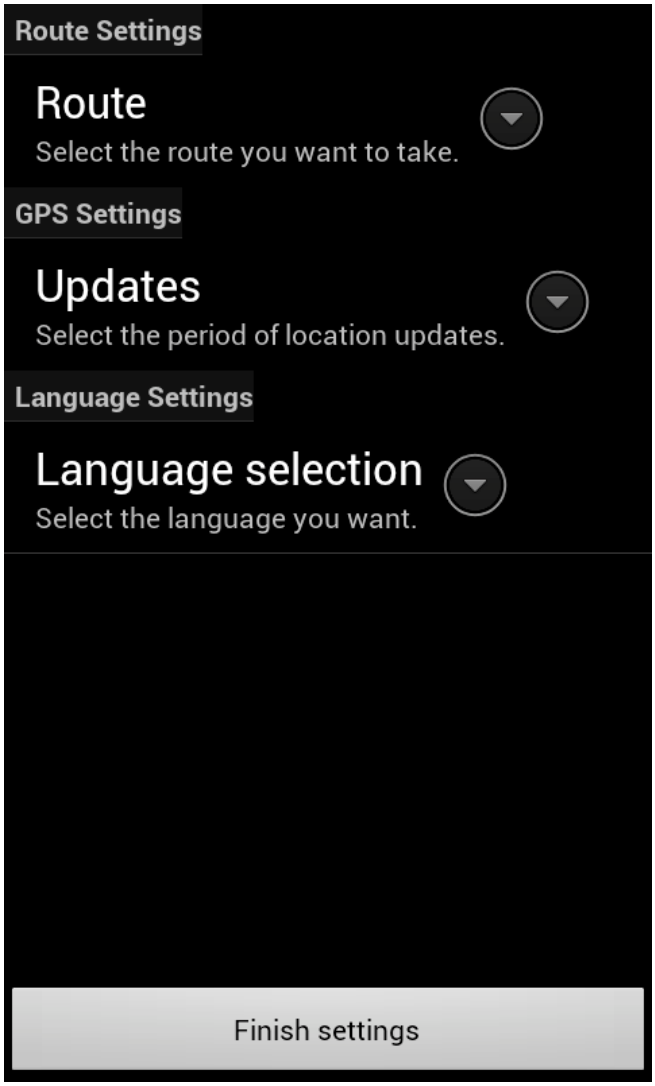

Ilustración 4 - Pantalla de configuración de la aplicación

En esta pantalla el usuario puede seleccionar los parámetros de configuración de la aplicación. Puede seleccionar el idioma de la aplicación, la ruta que desea realizar o el período de actualizaciones de la posición del usuario.

Una vez que ha realizado la configuración, pasa a una pantalla de explicación de la aplicación.

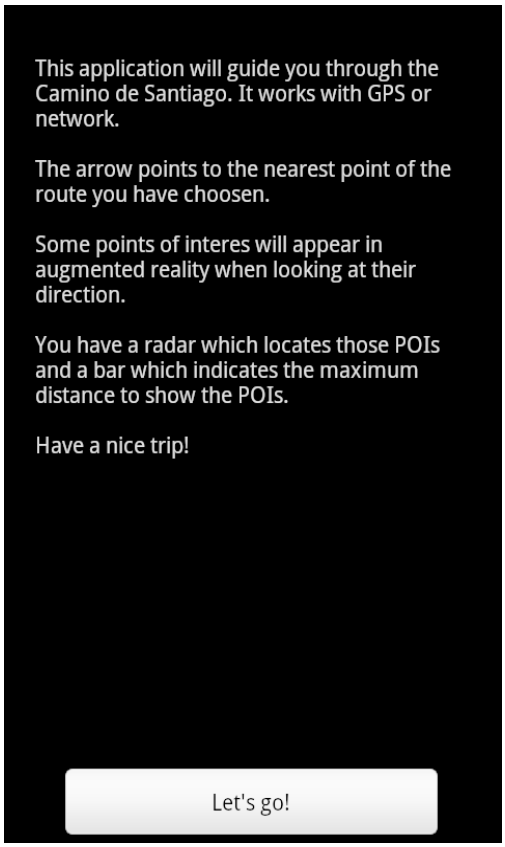

Ilustración 5 - Pantalla de explicación de la aplicación

Ahora la aplicación ya está preparada para ser ejecutada, ya que ya ha procesado todos los puntos de la ruta que el usuario ha elegido, por lo que pulsando el botón que se ve en la imagen empezaría el verdadero uso de la aplicación.

Ilustración 6 - Pantalla de muestra de la aplicación

3 – REQUISITOS

El objetivo principal de la aplicación es guiar al usuario mientras realiza el Camino de Santiago, bien mostrando un camino pintado en la pantalla, o señalando la dirección.

Como podemos observar en la siguiente figura, en los GPS convencionales tenemos ambas características, un camino pintado sobre la carretera que hemos de seguir, y una flecha que indica la dirección a la que estamos mirando en dicho momento, lo que muchas veces sirve para volver a situarnos cuando nos perdemos.

Ilustración 7 - GPS convencional

Lo comentado es la funcionalidad más básica de un GPS, marcar el camino a seguir, mostrando además diferentes datos como la velocidad a la que vamos, la distancia que nos queda, la hora a la que llegaremos al destino, etc.

Por lo tanto, dado que es la principal funcionalidad de la aplicación, intentar conseguir señalar el camino a seguir se convirtió en el primer objetivo. Y dentro de señalar el camino, necesitaba un objeto con que señalarlo.

Lo habitual dentro de los GPS es utilizar una flecha para indicar la dirección que debemos seguir, por lo que ese fue el objeto elegido para no romper con lo clásico.

Actualmente los dispositivos GPS han mejorado mucho en su interfaz y han evolucionado en sus características, tratando de incluir objetos en 3D en vez de objetos en 2D como era antes, haciéndolos más atractivos para el usuario, como se puede ver en la siguiente imagen.

Ilustración 8 - GPS moderno

Como se comprueba, la evolución visual es considerable, pero sigue sin ser lo suficientemente real. Este es el motivo de tratar de hacerlo en realidad aumentada, que resulte lo más real posible.

El primer paso fue buscar posibles aplicaciones que realizaran esto mismo en realidad aumentada. La primera que lo realizó y la más destacada actualmente es Wikitude, un GPS como cualquier otro que pinta el camino sobre la vista de la cámara, es decir, sobre las calles y carreteras del camino que estamos realizando.

Ilustración 9 - Aplicación GPS Wikitude

En la imagen se observa que el trabajo está muy bien conseguido y el objetivo de este proyecto no es igualarlo, principalmente porque es una aplicación encaminada a realizar los trayectos en coche, y este proyecto lo que pretende es servir de guía a la gente que realice el Camino de Santiago a pie.

Pero lo que si se puede intentar es mostrar los puntos de interes de una forma similar a la que lo hace Wikitude.

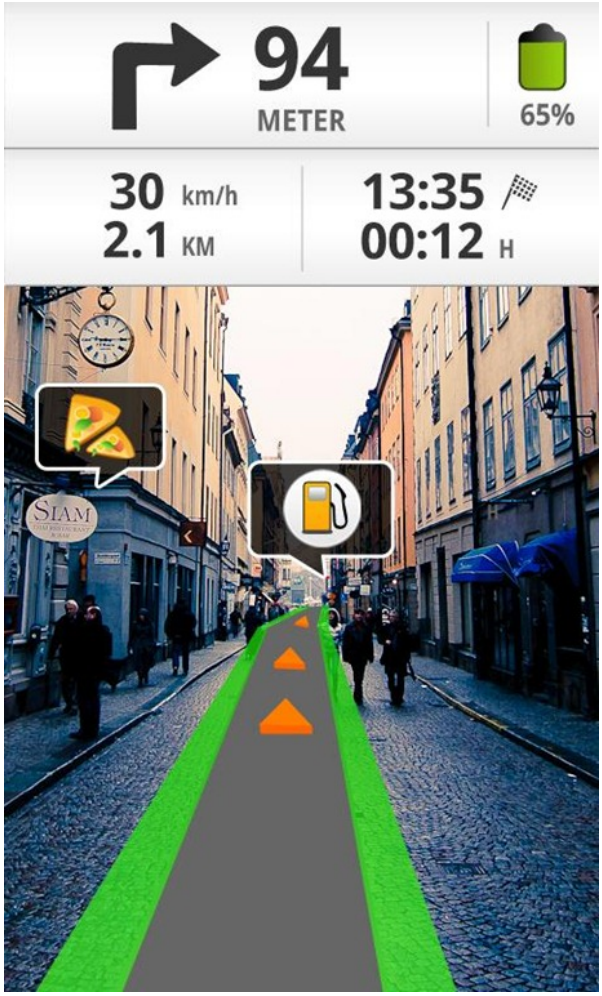

Ilustración 10 - Ejemplo Wikitude

Se observan distintos puntos de interés como un restaurante o una gasolinera, y la idea es esa, mostrar puntos de interés con diferentes iconos para indicar al usuario la presencia de elementos que rodean al Camino de Santiago realizado a pie como hostales, restaurantes, monumentos, etc.

Una vez definidos los elementos que vamos a dibujar sobre la vista de realidad aumentada, la flecha y los puntos de interés, hay que analizar las distintas tecnologías que existen para realizar dicha tarea.

Las tecnologías que podemos encontrar hoy en día para dibujar en Android no son muy variadas, y entre las que existen nos encontramos con Canvas y OpenGL ES, las más destacadas en este campo.

Canvas es una tecnología de Android que nos permite pintar objetos en dos dimensiones sobre un lienzo rectangular, que puede ser una superficie que nosotros queramos o directamente la pantalla del teléfono, que es lo que buscamos.

OpenGL ES es la otra tecnología que existe en Android para dibujar elementos en dos dimensiones, pero también en tres dimensiones, algo interesante si tenemos en cuenta que el elemento que se emplee para señalar el camino debe hacerlo en 360º.

Una vez analizadas ambas tecnologías, se decidió emplear las dos, Canvas para mostrar los puntos de interés en la pantalla dado que lo mejor es mostrarlos en dos dimensiones, y utilizar OpenGL ES para crear la flecha en tres dimensiones debido a que tiene que girar en todas direcciones según donde se encuentre mirando el usuario.

Los puntos de interés que se muestran al usuario se han realizado con Canvas, dado que verlos en dos dimensiones es lo más indicado y además dicha tecnología ofrece un mejor rendimiento cuando lo que dibujamos en el dispositivo no tiene que cambiar muchas veces por segundo.

Cada uno de los puntos de interés es un mapa de bits en el que unimos diferentes características, una imagen que pretende dar información a simple vista del tipo de punto de interés que se ha encontrado y un cuadro de texto en el que incluir más información, como el nombre del punto de interés, bien “Hostal X” o “Restaurante Y”, y la distancia a la que se encuentra dicho punto de interés.

Ilustración 11 - Punto de interés de la aplicación

En la imagen se muestra lo comentado anteriormente, el icono con el tipo de punto de interés y la información del nombre y la distancia a la que se encuentra.

Para realizar la flecha que indica la dirección que debemos seguir, se ha optado por diseñarla usando OpenGL ES [5], ya que la flecha debía representarse en tres dimensiones.

Para ello, OpenGL ES permite realizar el objeto 3D que nosotros queramos por código, definiendo una serie de vértices que forman unos triángulos, que a su vez forman el objeto elegido.

Otra forma de renderizar un elemento con OpenGL ES es a partir de un objeto creado con algún programa de diseño 3D, como puede ser 3DS MAX o Blender, y exportando el objeto diseñado en formato obj, que es un formato de archivo de definición geométrica aceptado por multitud de programas.

En dicho archivo se definen las posiciones de los vértices, la posición de las texturas, las normales, y las caras que representan cada polígono definido como una lista de vértices y vértices de textura.

Por tanto, el primer paso fue diseñar una flecha en tres dimensiones usando uno de estos programas. Dado que había manejado anteriormente 3DS MAX fue dicho programa el utilizado para crear la flecha.

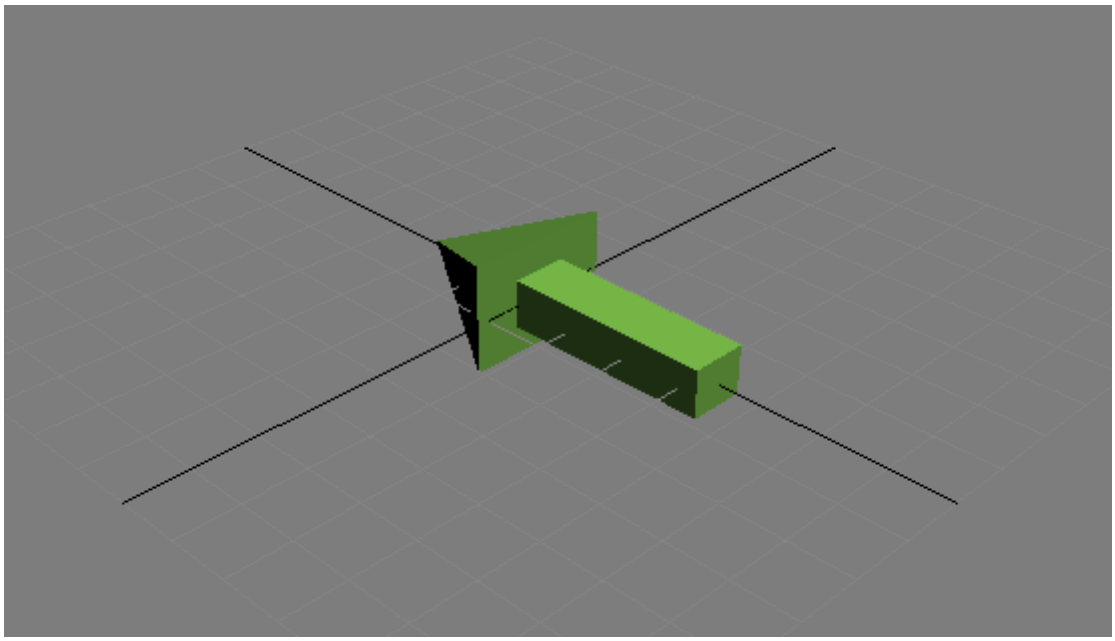

Ilustración 12 - Flecha 3D en 3DS Max

En la imagen se observa como es la flecha diseñada. El siguiente paso era obtener el archivo obj que OpenGL ES pudiera renderizar, pero la versión empleada del programa no permitía exportar archivos de este tipo, por lo que la solución fue exportar en otro tipo de archivo que Blender pudiera reconocer, y a su vez utilizar Blender para exportar a obj.

Una vez obtenido el archivo correspondiente, ya se pudo renderizar la flecha en el dispositivo con la tecnología OpenGL ES. Una vez renderizada la flecha y asignados los colores de cada cara, el resultado es el siguiente.

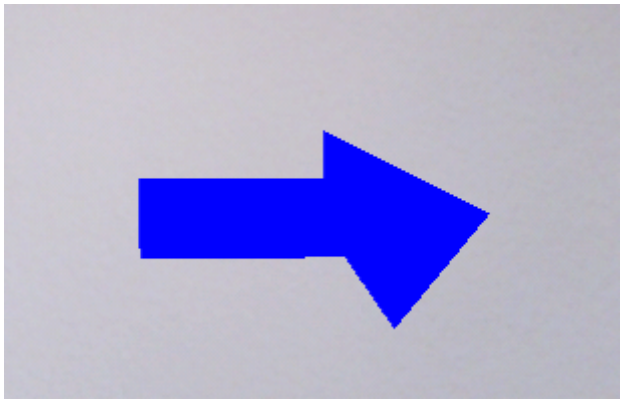

Ilustración 13 - Flecha 3D en la aplicación

Para finalizar, y después de todo lo comentado en este apartado se establecen dos casos de uso principales en la aplicación, seguir la ruta del Camino de Santiago y seguir los puntos de interés. Además, se añade un nuevo caso de uso puesto que el usuario puede realizar una configuración previa al primer uso de la aplicación.

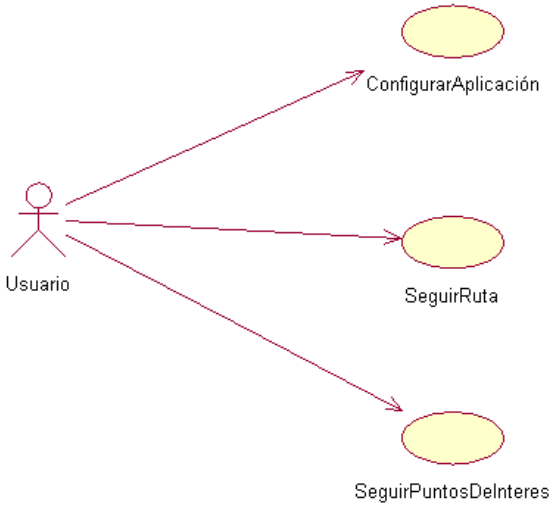

Ilustración 14 - Diagrama de casos de uso de la aplicación

4 – ANÁLISIS

Anteriormente se han comentado los requisitos funcionales que son primordiales para la finalidad de la aplicación, es decir, mostrar el camino y mostrar los puntos de interés. A continuación se van a analizar los elementos necesarios para llevar a cabo dicha funcionalidad.

4.1 – BBDD

Para conocer las distintas rutas del camino y los distintos puntos de interés es necesario almacenarlos de alguna forma para acceder a ellos en el momento en que sea necesario.

Existen distintas opciones como disponer de bases de datos donde almacenar los puntos, archivos gpx que son los que emplean los dispositivos GPS convencionales, documentos XML etiquetados de manera que posteriormente se pueda acceder a cada uno de los puntos de forma ordenada.

Después de analizar las diferentes opciones existentes, se ha decidido emplear distintas bases de datos para cada una de las rutas y para los distintos puntos de interés. La forma de manejar las bases de datos en Android es a través de SQLite. Previamente, hay que realizar un proceso de copia de la base de datos en el dispositivo, que en un principio se encuentra en la carpeta *assets* del proyecto y que tenemos que copiar en la partición *data* del teléfono.

Lo primero que hay que hacer antes de comenzar a trabajar con las bases de datos en Android, es registrarlas en el dispositivo para que podamos acceder posteriormente a sus contenidos. Para ello, antes de poder trabajar con una base de datos, debemos crear un objeto helper que permite crear, abrir o administrar una base de datos llamando al constructor de la clase SQLiteOpenHelper,

public SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

Al cual se le pasan el contexto en el que se encuentra la aplicación, el nombre que queremos dar a la base de datos o el que ya tiene, y la versión de la base de datos para ver si tiene que actualizarla o crear una nueva. Con este objeto sólo podremos manejar la base de datos con el nombre que hayamos indicado.

Una vez disponemos de este objeto, hay que realizar una comprobación previa para ver si ya disponemos de dicha base de datos en el teléfono. Si la base de datos no existe, quiere decir que tenemos que copiarla y ello lo conseguimos haciendo uso de buffers y leyendo byte a byte de nuestro archivo de base de datos.

Una vez hemos copiado la base de datos en el teléfono, ya podemos acceder a los datos que contenga. Este proceso de copia se realiza dos veces únicamente, una vez para la base de datos de la ruta que hayamos elegido, y otra vez para la base de datos de los puntos de interés. Las posteriores veces que utilicemos la aplicación, la comprobación simplemente dirá que ya tenemos las bases de datos y que podemos emplearlas.

Cuando la aplicación se encuentra en funcionamiento, se accede en dos momentos a su contenido, ambas veces es al crearse la actividad principal y se realiza llamando a dos funciones: *crearRuta( )* y *crearMarcadores( ).*

Dichas funciones realizan acciones similares, pero difieren en una cosa: ambos elementos se codifican de la misma forma pero los puntos de interés son interactivos y permiten ser pulsados, ofreciendo cierta información que aparece cuando se realiza dicha acción.

La información que se muestra es la que se encuentra almacenada en la base de datos de los puntos de interés a la que se accede de la siguiente forma: se obtiene un objeto *SQLiteDatabase* que se puede definir como sólo de lectura, y con el se efectúa una consulta a la base de datos, el resultado de la consulta se nos devuelve en un *Cursor* que debemos recorrer elemento a elemento creando cada uno de los marcadores.

La posibilidad de tener distintos puntos de interés hace que debamos almacenar su tipo en la base de datos, así como un identificador, su nombre, su latitud y longitud (coordenadas GPS que harán que aparezca en nuestro dispositivo dependiendo de nuestra localización) y su descripción, que será la información que aparecerá en un *Toast* cuando pulsemos un marcador en concreto. Un *Toast* es una vista flotante que aparece sobre la aplicación y que contiene un pequeño mensaje para el usuario. No pretende ser intrusivo sobre lo que esté realizando el usuario pero a su vez pretende conseguir que el usuario vea la información que se le quiere mostrar.

Una vez tenemos todos los marcadores creados, se tienen que presentar en la vista en la que nos encontramos. Esta vista se actualizará cada vez que se reciba una nueva posición GPS, se solicite una nueva distancia a la que mostrar los marcadores y cuando se inicie o reanude la aplicación.

Lo comentado anteriormente se aplica a los marcadores de los puntos de interés. Para los marcadores de la ruta se realizan las mismas acciones y se crean de igual forma, pero no son interactivos como los puntos de interés, por lo que las bases de datos de las rutas no necesitan ni nombre ni una descripción que mostrar. Por ello, la base de datos únicamente contiene un identificador, su latitud y su longitud.

Una vez tenemos registrados los puntos de la ruta, cada vez que se actualice nuestra posición GPS se ordenará la lista de marcadores de la ruta y el punto que se encuentre más cercano en línea recta a nuestra posición será el seleccionado para que la flecha 3D que hemos diseñado apunte en su dirección. A continuación se comentará como se ha conseguido la detección de la orientación a la que debe señalar nuestra flecha.

4.2 – DETECCIÓN DE LA ORIENTACIÓN

Para empezar hay que comentar que si queremos saber la localización exacta en la que nos encontramos debemos hacer uso de ciertas características de nuestro dispositivo, que son bien la señal GPS o la señal de la red del teléfono (Wifi o datos).

Para poder hacer uso de estas características del dispositivo, debemos agregar en el manifiesto de la aplicación ciertos permisos que el usuario debe aceptar cuando instala la aplicación. Sin estos permisos la aplicación fallaría ya que el sistema Android no nos permitiría utilizarlos. Estos permisos son:

android.permission.ACCESS_FINE_LOCATION (permite a la aplicación acceder a la señal GPS del dispositivo)
android.permission.ACCESS_CORSE_LOCATION (permite a la aplicación acceder a la señal Wifi/datos del dispositivo)

Con los permisos ya aceptados, necesitamos un objeto que nos avise cuando nuestra posición cambie. Este objeto se llama *LocationListener* y es el encargado de realizar las acciones pertinentes cuando nuestra localización cambie o algunas características del proveedor de la localización cambien (que el usuario active/desactive el proveedor yo que su estado cambie).

Después es necesario solicitar que nuestra actividad sea notificada periódicamente de dichos cambios en la localización, lo que se hace mediante la siguiente llamada:

public void requestLocationUpdates (String provider, long minTime, float minDistance, LocationListener listener)

Uno de los parámetros de la función es el proveedor de la información de la localización, tanto el GPS (*LocationManager.GPS_PROVIDER*) como la señal de red (*LocationManager.NETWORK_PROVIDER*). Recibir actualizaciones de la geolocalización puede ser un proceso costoso en cuanto al tiempo, por lo que si necesitamos conocer una posición rápidamente tenemos la posibilidad de hacer una llamada a otra función que nos daría la última localización conocida.

Otro de los parámetros es el mínimo intervalo de tiempo en milisegundos con el que queremos que se reciban las actualizaciones, y lo mismo con el tercer parámetro, el mínimo intervalo de distancia en metros a la que queremos que se actualicen las posiciones.

Se puede controlar la frecuencia de las notificaciones empleando estos dos últimos parámetros: si el mínimo tiempo elegido es mayor que cero, el *LocationManager* puede descansar durante ese período de tiempo para preservar la energía del teléfono, ya que es un proceso que consume mucha. Si la mínima distancia es mayor que cero, una localización será señalizada sólo si el dispositivo se mueve en la distancia elegida.

Una vez hemos registrado el proveedor del que queremos recibir las actualizaciones, tendremos que comprobar si se encuentra operativo, motivo por el cual se registran ambos proveedores, el de la señal GPS y el de la señal de red. El primero de los proveedores que se comprueba es el del GPS, dado que es el más preciso de los dos. Si dicho proveedor se encuentra activado, se intenta tomar la última localización que haya detectado con la función:

public Location getLastKnownLocation (String provider)

A la que pasamos el proveedor que elijamos y que nos devuelve la última localización conocida.

En caso de no estar activado el GPS del dispositivo se intentaría tomar la última posición conocida por la señal de red mediante la misma función, y en caso de que tampoco estuviese activado, generaríamos una localización GPS que en esta ocasión sería por defecto la primera posición del Camino de Santiago de la siguiente forma:

```
public static final Location hardFix = new Location("ATL");
    static {
        hardFix.setLatitude(43.009547);
        hardFix.setLongitude(-1.319764);
        hardFix.setAltitude(1);
    }
```

Una vez hemos encontrado nuestra geolocalización es tiempo de saber hacia en qué dirección nos encontramos mirando con nuestro dispositivo y para ello es necesario hacer uso de otras características a parte de las comentadas anteriormente, que son: el acelerómetro y el sensor magnético.

Un acelerómetro se define como un instrumento para medir la velocidad de variación de velocidad con respecto a la magnitud o la dirección, es decir, movimientos en la vertical o la horizontal del dispositivo. El sensor magnético mide el vector del campo magnético de la Tierra, representado en el sistema de coordenadas del dispositivo, vector el cual apunta al polo norte magnético.

Para empezar se toma de la lista de los sensores del dispositivo los dos comentados, y se registran de forma que la clase que renderizará la flecha 3D que indica el camino para que sea ésta la que reciba las actualizaciones de ambos sensores de forma continua.

```
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
List<Sensor> listSensors = mSensorManager
       .getSensorList(Sensor.TYPE_ACCELEROMETER);
if (listSensors.size() > 0) {
       mSensorManager.registerListener(compassRenderer,
       listSensors.get(0), SensorManager.SENSOR_DELAY_UI);
}

listSensors = mSensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
if (listSensors.size() > 0) {
       mSensorManager.registerListener(compassRenderer,
       listSensors.get(0), SensorManager.SENSOR_DELAY_UI);}
```

Lo siguiente es ya esperar a recibir las notificaciones de ambos sensores y realizar los cálculos pertinentes para hacer que la flecha apunte a la posición GPS que nosotros queramos. En cada ocasión en que vayamos a dibujar la flecha 3D, debemos realizar lo siguiente:

Lo primero es llamar a la función del SensorManager que nos devuelve una matriz de rotación en función de los valores que nos hayan dado los sensores acelerómetro y magnético:

```
SensorManager.getRotationMatrix(rotationMatrix, null, mAccelerometerValues,
mMagneticValues);
```

En el primer parámetro de la función obtendremos la matriz de rotación transformando un vector del sistema de coordenadas del dispositivo al sistema de coordenadas del mundo que se define como una base ortonormal directa [7], donde:

- X se define como el vector producto Y*Z (tangencial al suelo en la localización actual del dispositivo y que apunta al Este).
- Y es tangencial al suelo en la localización actual del dispositivo y apunta hacia el Polo Norte magnético.
- Z apunta hacia el cielo y es perpendicular al suelo en la localización actual del dispositivo.

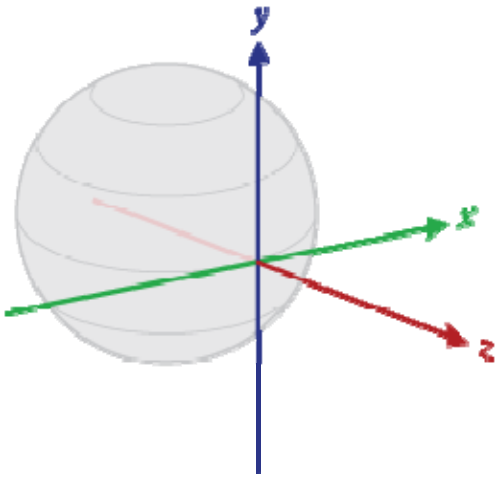

Ilustración 15 - Sistema de coordenadas del Mundo

Lo siguiente es tomar la matriz de rotación que hemos obtenido anteriormente, y convertirla al sistema de coordenadas del dispositivo, obteniendo su orientación que en este caso será en modo apaisado, elegido por ser la forma en la que el usuario aprovecharía mejor las características de la aplicación.

```
SensorManager.remapCoordinateSystem(rotationMatrix,       SensorManager.AXIS_Y,
SensorManager.AXIS_MINUS_X, remappedRotationMatrix);
SensorManager.getOrientation(remappedRotationMatrix, orientacion);
```

Una vez computada la orientación del dispositivo, disponemos de ciertos valores acerca del nuevo eje de coordenadas, que como se observa es distinto al obtenido anteriormente.

- X se define como el vector producto Y*Z (tangencial al suelo en la localización actual del dispositivo y que apunta al Oeste).
- Y es tangencial al suelo en la localización actual del dispositivo y apunta hacia el Polo Norte magnético.
- Z apunta hacia el centro de la Tierra y es perpendicular al suelo en la localización actual del dispositivo.

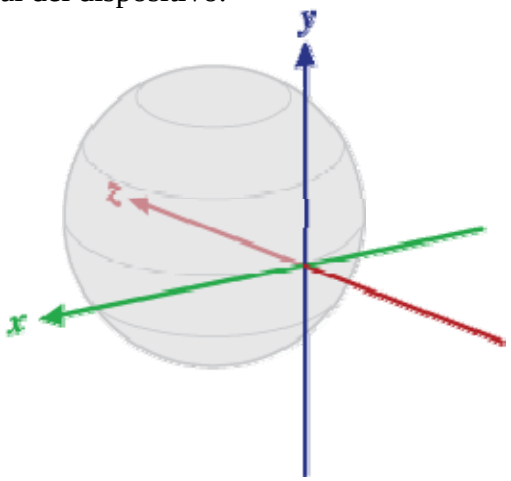

Ilustración 16 - Sistema de coordenadas del dispositivo Android

`orientacion[0]` → azimuth, rotación alrededor del eje Z
`orientacion[1]` → pitch, rotación alrededor del eje X
`orientacion[2]` → roll, rotación alrededor del eje Y

En este punto es donde debemos tomar la geolocalización actual del usuario **[** 3**]**, ya que debemos corregir el azimuth obtenido con la orientación por tener uno de los ejes apuntando al Polo Norte Magnético y no al verdadero Norte.

Para ello, primero se tiene que pasar a grados el valor obtenido del azimuth ya que lo obtenemos en radianes, crear un campo geomagnético según nuestra localización y hallar su declinación, que es la declinación de la componente horizontal del campo magnético respecto al verdadero Norte. Una vez restado al azimuth dicho valor, ya conseguimos obtener la dirección a la que se encuentra el verdadero Norte y no el Polo Norte Magnético.

```
this.loc = ARData.getCurrentLocation();
float azimuth = orientacion[0];
azimuth = azimuth * 360 / (2 * (float) Math.PI);
GeomagneticField geoField = new GeomagneticField(
            Double.valueOf(this.loc.getLatitude()).floatValue(),
            Double.valueOf(this.loc.getLongitude()).floatValue(),
            Double.valueOf(this.loc.getAltitude()).floatValue(),
                        System.currentTimeMillis());
azimuth -= geoField.getDeclination();
```

Ahora que ya tenemos el valor del azimuth corregido y en grados, podemos continuar con la indicación del camino. Hasta ahora nos hemos centrado en nuestra localización, pero a continuación hay que direccionar el rumbo desde nuestra localización hasta el punto de la ruta que sea.

Anteriormente ya hemos registrado todos los puntos de la ruta del Camino de Santiago que ha elegido el usuario y lo que hay que hacer ahora es seleccionar como siguiente destino el punto que más cerca se encuentre de nosotros. Para ello hacemos lo siguiente:

```
List<Marker> markers = ARData.getRouteMarkers();
Marker m = markers.get(0);
double[] latlong = m.getLatLong();
```

Lo que hemos hecho ha sido recoger los puntos de la ruta en una lista de marcadores al igual que haríamos con los puntos de interés de la ruta, y tomar el primero de ellos a través de la función *getRouteMarkers()*, que nos ordena la lista de puntos en función de la distancia a la que se encuentren, de menor a mayor.

Posteriormente creamos un nuevo punto de localización a partir de nuestra localización, ya que el API de Android no nos permite crear una localización vacía, y cambiar la latitud y la longitud por las del punto seleccionado.

```
target = new Location(this.loc);
target.setLatitude(latlong[0]);
target.setLongitude(latlong[1]);
```

Ahora que ya tenemos localizado el siguiente punto al que debemos dirigirnos, tenemos que ver el rumbo que debemos tomar para dirigirnos a él. Para ello existe una función que lo calcula automáticamente entre dos localizaciones gracias al API de Android.

```
float bearing = this.loc.bearingTo(target);
if (bearing < 0) {
       bearing = bearing + 360;
}
```

Dicha función nos da el rumbo inicial aproximado en grados del Este al verdadero Norte en el camino más corto entre dichos puntos. Si el rumbo es negativo debemos sumarle 360º para convertirlo en positivo. Con esto no es suficiente, ya que aún no estamos dirigiéndonos directamente al punto, para ello, es necesario realizar una última operación, restarle dicho rumbo al azimuth obtenido anteriormente.

```
float grados = azimuth - bearing;
if (grados < 0) {
       grados = grados + 360;
}
```

De nuevo tenemos que comprobar que el resultado no sea negativo, y en caso contrario sumarle 360º.

Ahora sí disponemos de los grados que necesitamos girar en nuestro rumbo para llegar al punto siguiente de la ruta dependiendo no sólo de nuestra posición, si no de la dirección en la que estemos enfocando el dispositivo. Lo único que nos queda ahora es aplicar dichos grados a la flecha, que siempre está en principio apuntando al frente, y que se realiza de forma sencilla con una operación.

```
gl.glRotatef(grados, 0.0f, 1.0f, 0.0f);
```

Con esta función podemos rotar un vector cualquiera en los grados que queramos y en el eje que queramos. En este caso, lo que hacemos es rotar los grados obtenidos restando al azimuth el rumbo y aplicándolos al eje Y de la flecha, que dentro del dispositivo en modo apaisado sería el siguiente:

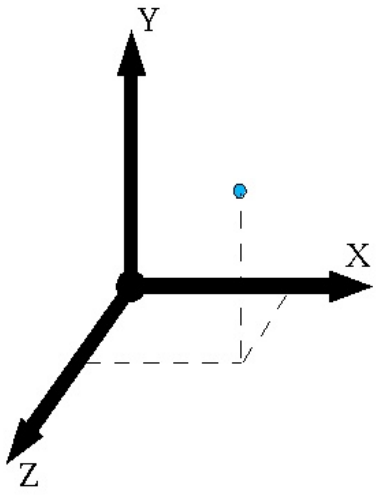

Ilustración 17 - Sistema de coordenadas de OpenGL ES

Lo último a comentar de la renderización de la flecha es que se reciben datos de los sensores muy fácilmente, al mínimo movimiento del dispositivo, y en consecuencia se debe dibujar una nueva flecha que apunte correctamente al objetivo. Para minimizar estos movimientos de le flecha y que no se observe bruscamente, se aplica a los datos de los sensores un filtro de paso bajo para suavizar su movimiento. Ello se realiza de la siguiente forma:

```
switch(event.sensor.getType()) {
       case Sensor.TYPE_ACCELEROMETER:
       mAccelerometerValues[0]=(mAccelerometerValues[0]*8+event.values[0])*0.11
112f;
       mAccelerometerValues[1]=(mAccelerometerValues[1]*8+event.values[1])*0.11
112f;
       mAccelerometerValues[2]=(mAccelerometerValues[2]*8+event.values[2])*0.11
112f;
       break;
}
```

Con esto lo que conseguimos es que los nuevos datos de los sensores influyan en una novena parte en la renderización de la flecha. Lo mismo se hace con el sensor magnético.

5 – DISEÑO E IMPLEMENTACIÓN

En este apartado se van a explicar las diferentes clases que se han tenido que implementar para la consecución del proyecto. Se va a comenzar con una explicación general de las clases que intervienen en él y posteriormente se profundizará en las más interesantes de entre ellas.

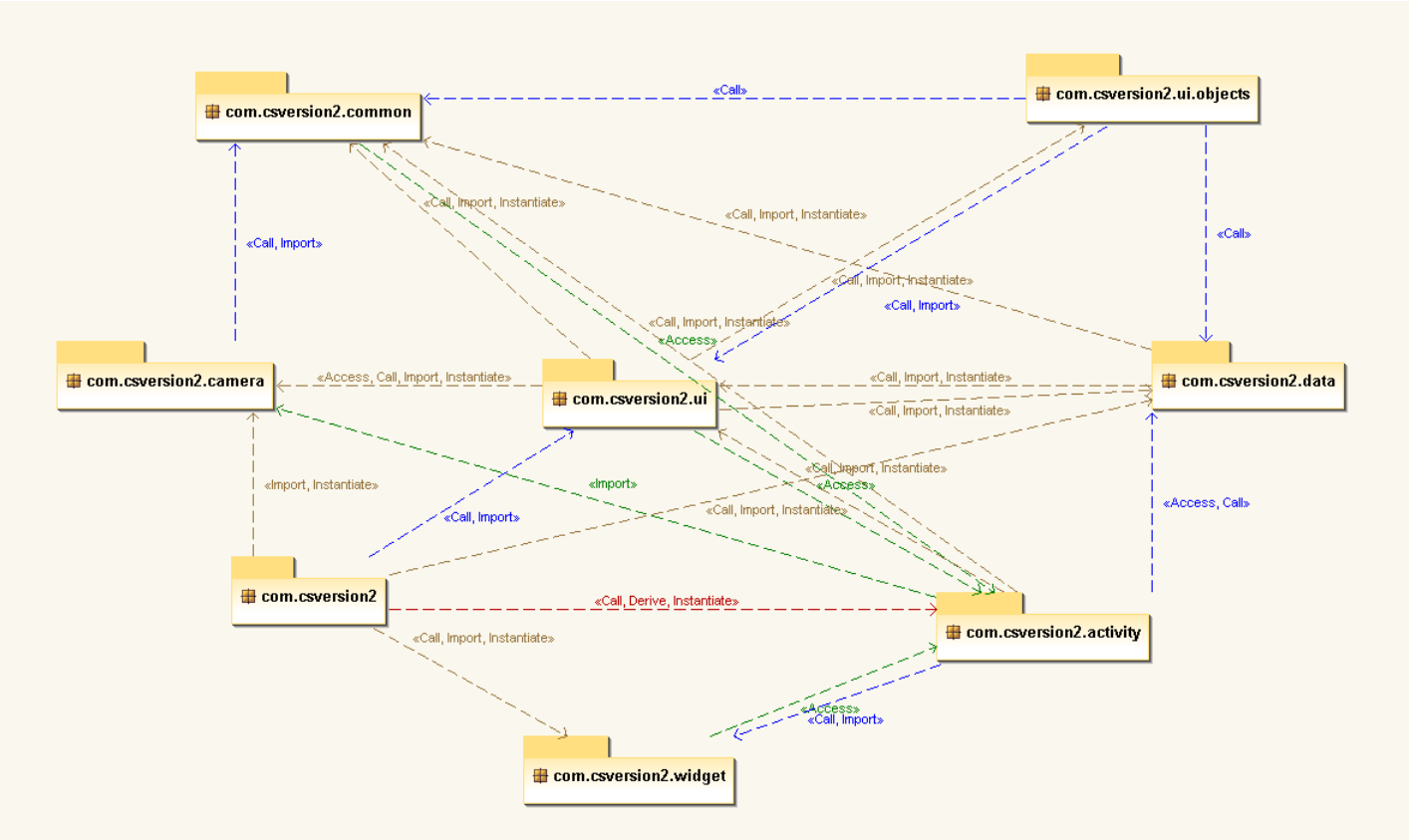

Ilustración 18 - Diagrama de paquetes de la aplicación

En la imagen anterior vemos la interacción entre los diferentes paquetes de clases que componen la aplicación. A continuación se van a explicar las clases más importantes que intervienen en cada uno de los paquetes de la misma.

Vamos a comenzar por la parte más visual de la aplicación, los gráficos que se muestran mientras nos encontramos usándola. Son varios los paquetes donde se definen los variados elementos que los componen, entre otras son *objects*, *ui* y *widget*.

5.1 - Objects

El paquete *objects* contiene multitud de clases, las cuales heredan todas de la clase *PaintableObject*, y como su propio nombre indica, sirve para definir los elementos que se pueden dibujar en la pantalla de nuestro dispositivo.

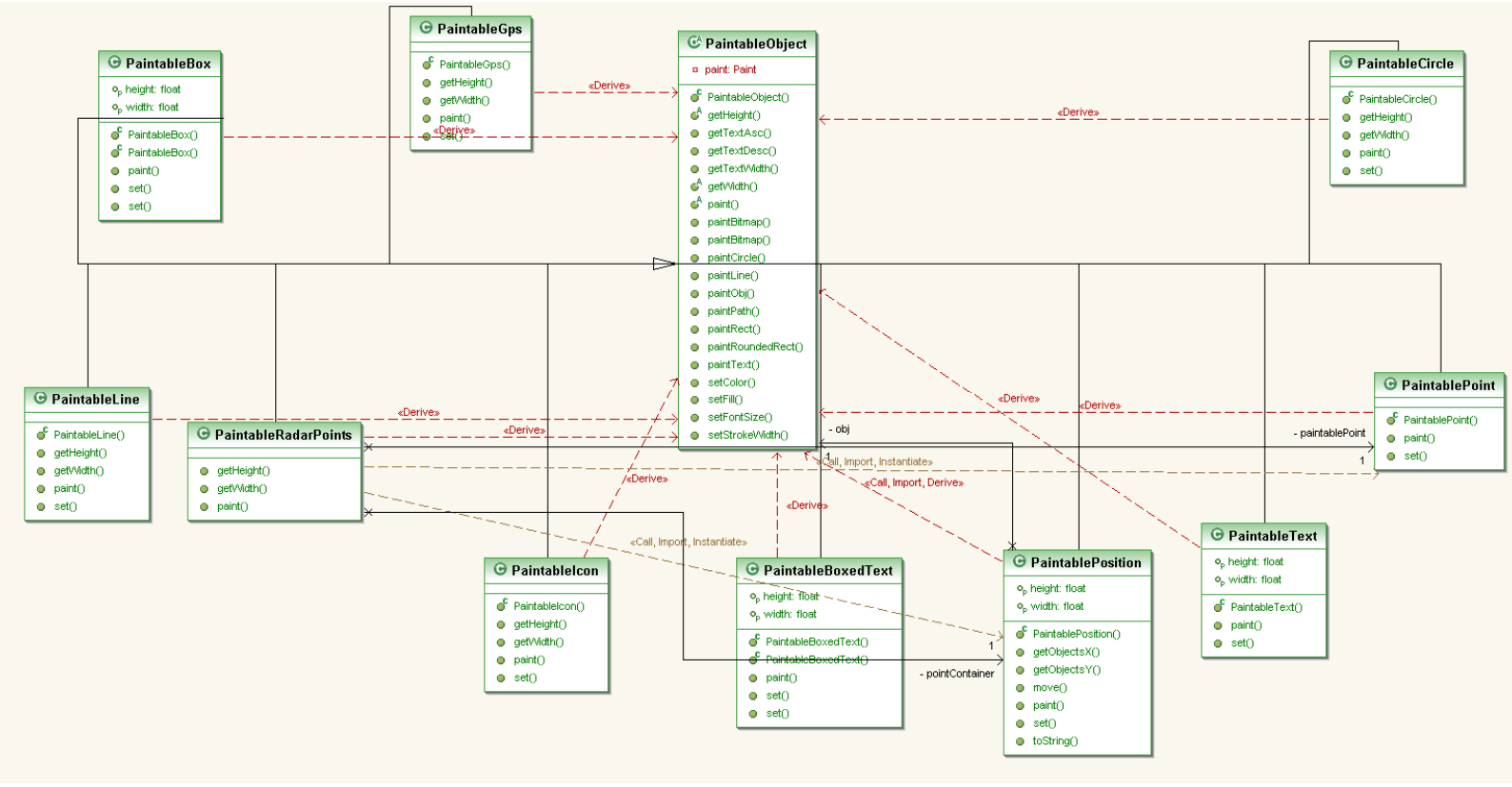

Ilustración 19 - Diagrama de clases del paquete Objects

En las clases de este paquete se definen los distintos tipos de elementos que podemos dibujar, como los puntos dentro del radar o las cajas de texto donde se muestra la información de los puntos de interés.

5.2 – Ui

El siguiente paquete, *ui*, contiene las definiciones de los marcadores que utilizamos bien para señalar el camino o para mostrar los puntos de información, así como el radar que es donde colocamos los puntos del paquete *objects*.

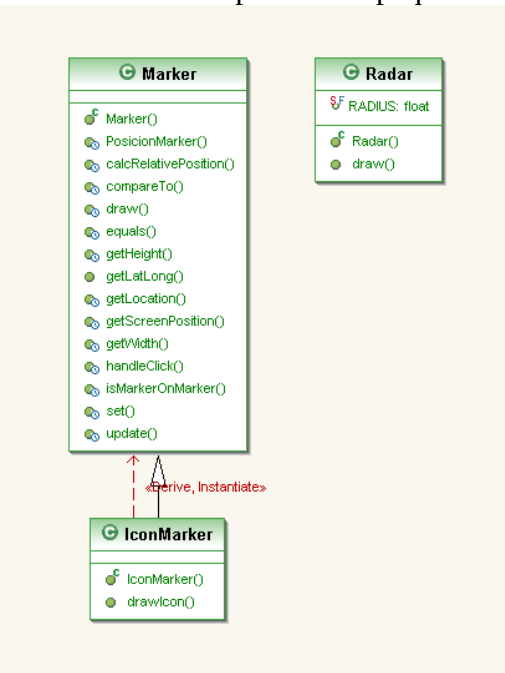

Ilustración 20 - Diagrama de clases del paquete Ui

Como vemos la clase que utilizamos para definir nuestros puntos de interés es *IconMarker*, que hereda de *Marker*, y la única diferencia existente entre ambas clases es la posibilidad que nos ofrece la primera de ellas de mostrar una imagen asociada al marcador en cuestión. Una de las funciones más interesantes que contiene la clase *Marker* es:

```
public synchronized void calcRelativePosition(Location location)
```

Cuya función es calcular la posición que un marcador debería ocupar en la pantalla del dispositivo a partir de la localización geográfica del usuario y dependiendo también de la dirección en la que nos encontremos mirando, por lo que si nos encontramos mirando al Norte y hay un punto de interés en el Sur, no nos aparecerá en la pantalla del dispositivo pero sí aparecerá en el radar debajo de nuestra posición. Otras funciones también importantes son las que dibujan dichos elementos en la pantalla, que son:

```
protected synchronized void drawIcon(Canvas canvas)
private synchronized void drawText(Canvas canvas)
```

Como se puede suponer, dichas funciones lo que hacen es pintar el icono del punto de interés en cuestión y la información que mostramos al usuario antes de que pulse dicho marcador, su nombre y la distancia a la que se encuentra el punto de interés.

5.3 - Widget

El último paquete de clases que interviene en la parte gráfica es *widget*. Estas clases son las encargadas de dibujar el último elemento que falta, la barra que marca la distancia a la que queremos ver los puntos de interés.

Es interactiva en la medida en que el usuario desee, ya que es él el que tiene que decidir a qué distancia quiere ver los puntos, desde 0 metros hasta 20 kilómetros de distancia.

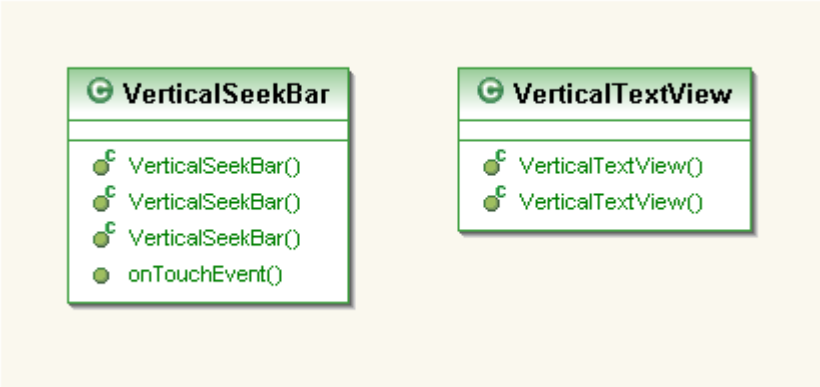

Ilustración 21 - Diagrama de clases del paquete Widget

Los paquetes de clases comentados son los encargados de manejar la parte gráfica de la aplicación, pero esta parte no podría hacer nada sin la parte funcional. Del resto de paquetes que se encargan de ello, destacan tres: *activity*, *data* y *csversion2*.

El siguiente paquete a comentar es *activity*.

5.4 - Activity

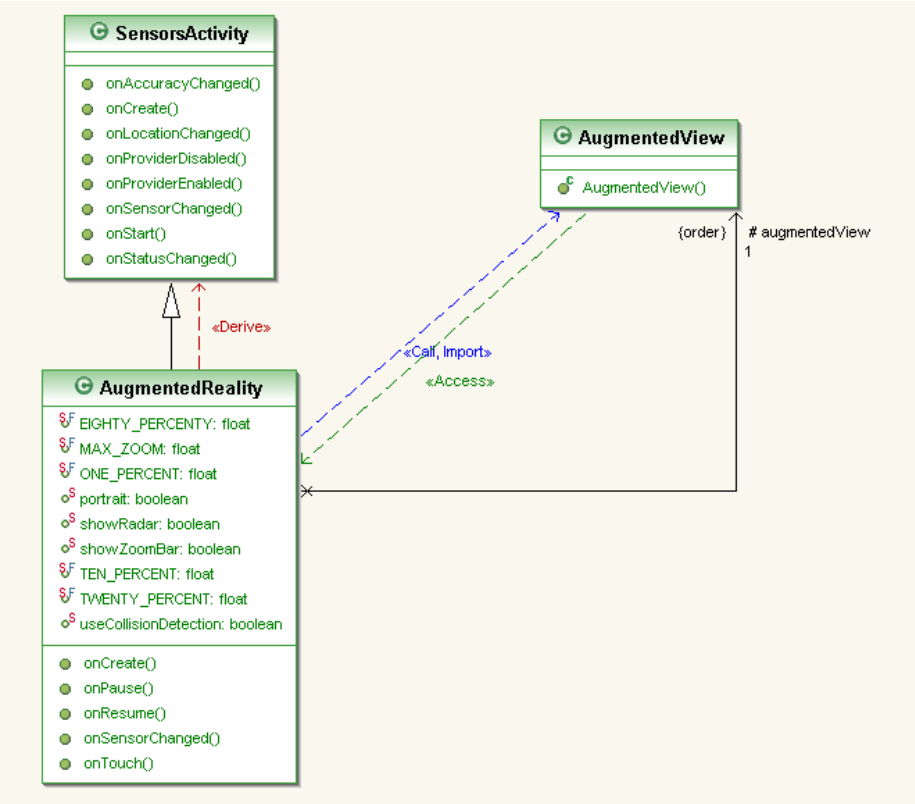

Ilustración 22 - Diagrama de clases del paquete Activity

A simple vista la clase más sencilla de las tres parece ser *AugmentedView*, pero es la encargada de dibujar en pantalla todos los elementos explicados en las anteriores clases, que forman la realidad aumentada de la aplicación.

La clase *AugmentedReality* hereda de *SensorsActivity* y está diseñada para unir el funcionamiento de la clase *AugmentedView*, es decir, los elementos del radar y los puntos de interés, y las clases del paquete *widget*, que definían la barra vertical con la que indicar la distancia a la que mostrar los puntos de interés tanto en la pantalla como en el radar.

Por último, la clase *SensorsActivity* se encarga de procesar toda la información de los sensores magnéticos y acelerómetro, así como obtener la localización geográfica del usuario. Para ello, registra los listener como ya hemos comentado en apartados anteriores y pide recibir actualizaciones de dichos elementos. Con la información recibida tanto de unos como de otros realiza las llamadas pertinentes para actualizar los elementos que se deben mostrar en la pantalla del dispositivo.

5.5 - Data

Otro paquete importante dentro de la aplicación es *data*.

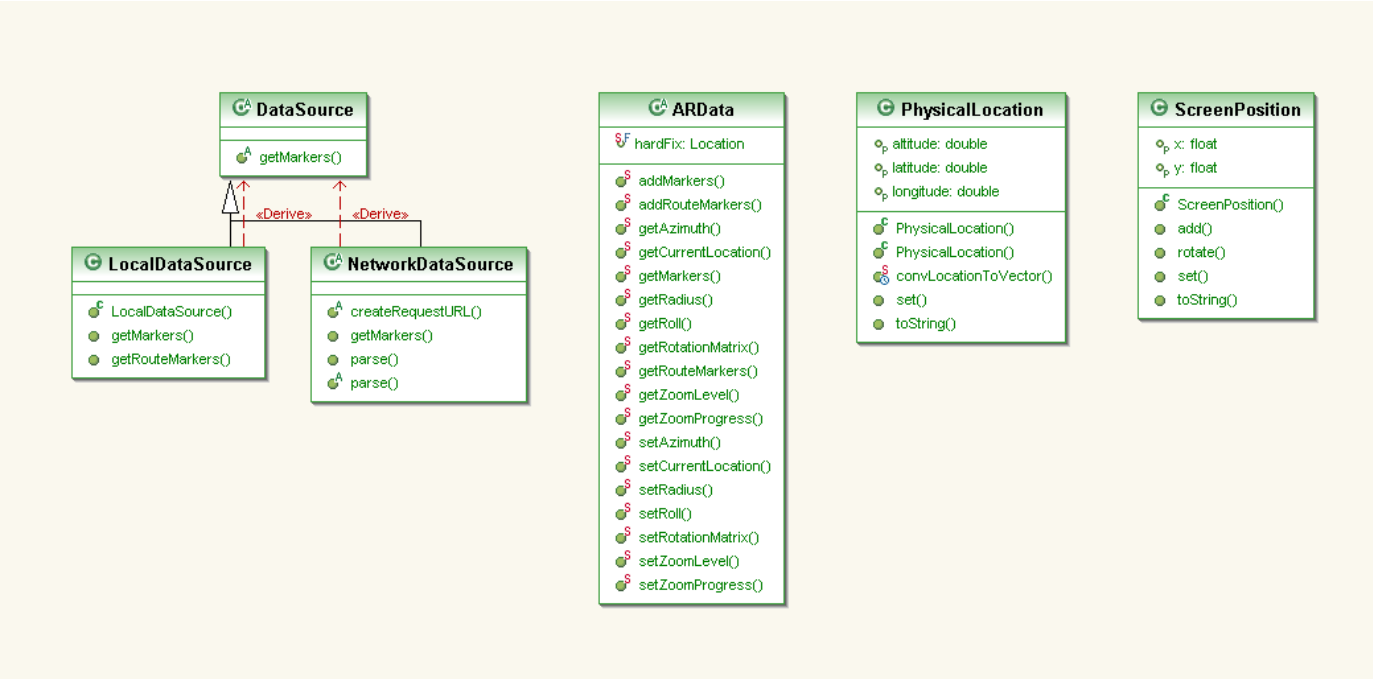

Ilustración 23 - Diagrama de clases del paquete Data

Este paquete de la aplicación contiene las clases encargadas de guardar la información sobre los puntos de interés y las rutas del camino. Como ya hemos comentado anteriormente, es gracias a las funciones *getMarkers()* y *getRouteMarkers()* de la clase *LocalDataSource* con las que conseguimos almacenar los distintos puntos.

Empleamos esta clase porque nuestra fuente de datos la tenemos en las bases de datos de las rutas y los puntos de interés, pero si tuviéramos los datos en servidores externos a la aplicación, podríamos emplear la clase *NetworkDataSource* para realizar el mismo trabajo que con las bases de datos locales.

Luego tenemos la clase *ARData*, que es una de las más importantes de la aplicación y es empleada por casi todo el resto de los paquetes de la aplicación. Se encarga tanto de administrar los puntos de la ruta y los puntos de interés como de guardar la información concerniente a la localización geográfica del usuario.

5.6 – CSVersion2

Por último, nos encontramos con el paquete de clases que guarda las clases más importantes del proyecto, y que son las que crean las actividades y vistas que va a ver el usuario, mostrándole la información idónea en cada momento. Es el paquete *csversion2*.

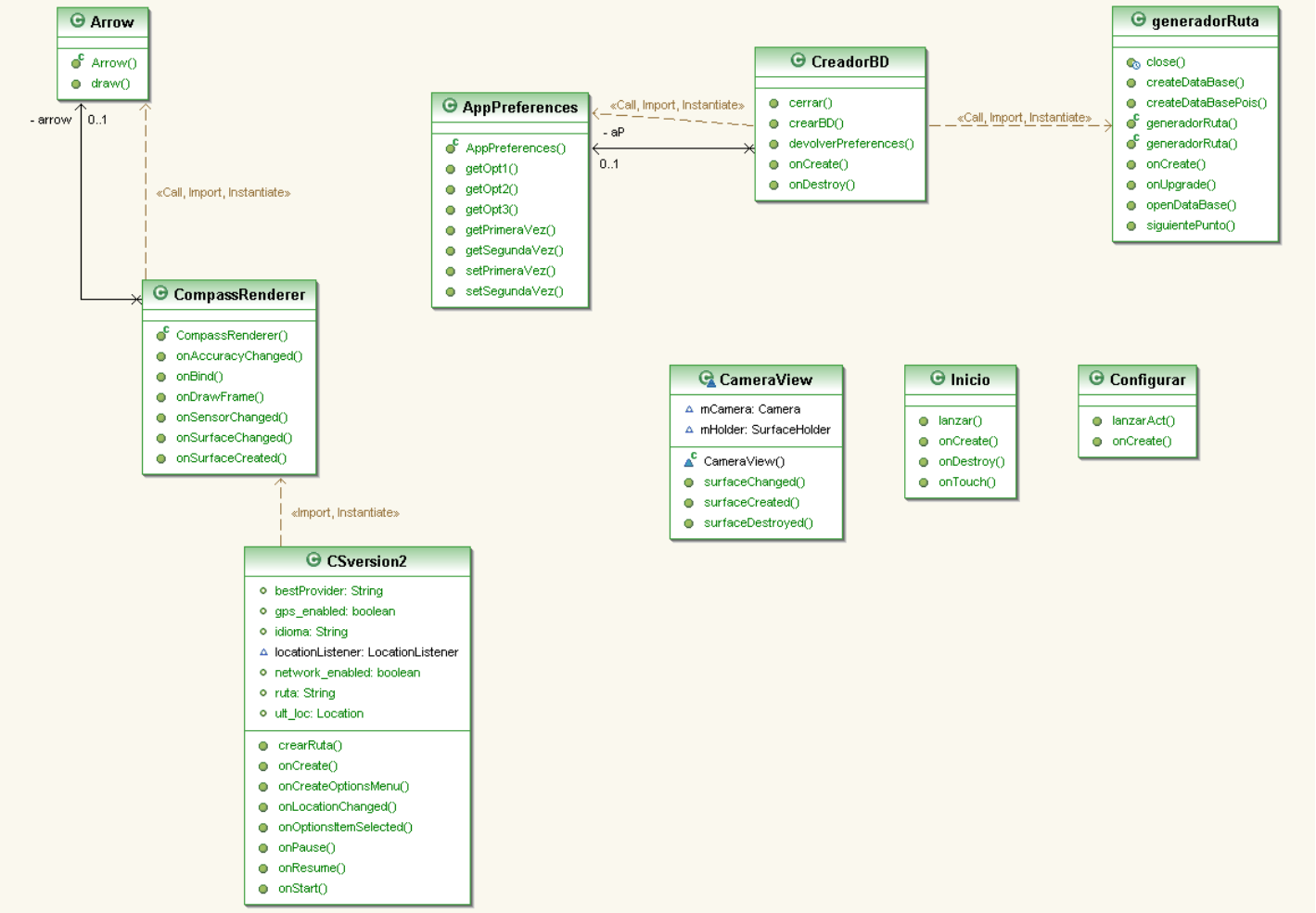

Ilustración 24 - Diagrama de clases del paquete CSVersion2

Las clases de este paquete son fundamentales para el funcionamiento de la aplicación, ya que controlan todos los procesos que intervienen en el mismo. Vamos a ir comentando una a una todas las clases en el orden en que funciona la aplicación.

La primera de ellas es la clase *Inicio*. Esta clase únicamente presenta una vista con un fondo de pantalla y un mensaje de texto que indica al usuario que pulsando la pantalla puede proceder a ejecutar la aplicación.

Una vez el usuario pulsa la pantalla, se ejecuta automáticamente la función *lanzar()* transcurrido un segundo. Esta función lo que hace es finalizar la actividad que se estaba ejecutando y lanzar una nueva, en este caso la clase *CreadorBD*.

Esta nueva clase es la que realiza la comprobación de si es la primera vez que ejecutamos la aplicación o no. Esto podemos saberlo gracias a una clase que nos ofrece el API de Android y cuya función es guardar todo tipo de preferencias que requiera nuestra aplicación, y se llama *SharedPreferences*.

Su funcionamiento consiste en guardar un archivo XML en una carpeta en la partición interna del dispositivo a la que no podemos acceder de forma manual por lo que está bien protegido. Este archivo adquiere un nombre automáticamente compuesto por el nombre del paquete principal de la aplicación y continuado por "*_preferences.xml*", por lo que en este caso nuestro archivo se llama *com.csversion2_preferences.xml*, y está estructurado de la siguiente forma:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="Option 3">ENG</string>
<string name="Option 1">NOR</string>
<string name="Option 2">5m</string>
<string name="primeraVez">no</string>
</map>
```

Como se observa, por cada preferencia que hayamos querido guardar se escribe una entrada en el archivo de preferencias. Cada una de las entradas tiene un nombre, que es el que le hemos dado nosotros y que emplearemos internamente en nuestro código para acceder a los mismos, y la etiqueta también dada por nosotros, que sirve de indicación sobre la preferencia que se ha seleccionado.

Para acceder a estas preferencias, hemos creado una clase propia llamada *AppPreferences* para hacer más sencillo su manejo. En este caso, como estamos intentando ver si es la primera vez que hemos iniciado la aplicación, intentaríamos acceder a la preferencia "primeraVez" de la siguiente forma:

```
public String getPrimeraVez() {
      return appSharedPrefs.getString("primeraVez", "si");
}
```

Con la función *getString(preferencia, valor_predeterminado)* lo que conseguimos es que si no existe ese valor almacenado en el fichero XML, devuelva lo que contiene *valor_predeterminado*, sino devolvería el valor almacenado en el fichero. Como es la primera vez que hemos iniciado la aplicación, no disponemos siquiera del archivo de preferencias, y mucho menos de la preferencia "primeraVez", por lo que se nos devolverá el valor "si" y procederemos a actualizar su valor, haciéndolo de la siguiente forma:

```
public void setPrimeraVez(String text) {
      prefsEditor.putString("primeraVez", text);
      prefsEditor.commit();
}
```

Ahora sí podemos proceder con el funcionamiento de la aplicación, cuyo siguiente paso será permitir al usuario que configure la aplicación como desee.

La clase que se encarga de realizar la configuración es *Configurar*. Esta actividad cambia de nuevo la vista que se ve en el dispositivo, y nos presenta una pantalla de configuración. Esta vista trabaja a través de una clase especial del API de Android, cuyo funcionamiento únicamente es, a partir de un archivo XML, mostrar una serie de opciones seleccionables al usuario. Este archivo se estructura de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
    <PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory android:title="Route Settings">
            <ListPreference android:title="Route" android:summary="Select the
route you want to take." android:key="Option 1" android:dialogTitle="Route"
android:entryValues="@array/routecode" android:entries="@array/route"/>
        </PreferenceCategory>
        <PreferenceCategory android:title="GPS Settings">
            <ListPreference android:dialogTitle="Period of updates"
android:title="Updates" android:summary="Select the period of location
updates." android:key="Option 2" android:entryValues="@array/timecode"
android:entries="@array/time"/>
        </PreferenceCategory>
        <PreferenceCategory android:title="Language Settings">
            <ListPreference android:entries="@array/lang"
android:entryValues="@array/langcode" android:dialogTitle="App's language"
android:key="Option 3" android:title="Language selection"
android:summary="Select the language you want."/>
        </PreferenceCategory>
    </PreferenceScreen>
```

El nodo más simple del archivo se llama *ListPreference*, y es con el cual indicamos las posibles opciones que puede seleccionar el usuario. Estas opciones no se encuentran directamente en este mismo archivo, sino que se referencian a otro archivo de la siguiente forma: `"@array/lang"`. En este caso, la opción configurable del idioma, muestra sus entradas a través del archivo “lang.xml”, que se estructura así:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
   <string-array name="lang">
      <item>English</item>
      <item>Spanish</item>
   </string-array>
   <string-array name="langcode">
      <item>ENG</item>
      <item>SPA</item>
   </string-array>
</resources>
```

El primer grupo de entradas son lo que verá el usuario a la hora de configurar la aplicación, y el segundo grupo será lo que se guarde en el archivo de preferencias. Por ello, en el archivo de preferencias de ejemplo mostrado anteriormente, la preferencia de nombre “Option 3” tiene como valor “ENG”, ya que el usuario ha seleccionado el idioma inglés para utilizar la aplicación, y queda codificado como tal en el archivo de preferencias.

Cuando el usuario ha configurado la aplicación, puede pulsar el botón de finalizar y en ese momento volvemos a la anterior actividad, ya que no la hemos finalizado antes de lanzar la configuración. En esta ocasión, la preferencia "primeraVez" ya no es "si" por lo que no presentará de nuevo la pantalla de configuración, sino que pasará directamente a crear las bases de datos de rutas y puntos de interés. La función que se encarga de ello se llama *crearBD()*, y realiza lo siguiente:

```
public void crearBD() {
	String ruta = this.AP.getOpt1();
	if(ruta.equals("NOR")){
		ruta = "RutaCaminoNorte.db";
	}
	else if(ruta.equals("NORP")){
		ruta = "RutaCaminoNortePrimitivo.db";
	}
	else if(ruta.equals("FRA")){
		ruta = "RutaCaminoFrances.db";
	}

	String lang = this.AP.getOpt3();
	if(lang.equals("ENG")){
		lang = "poisIngles.db";
	}
	else if(lang.equals("SPA")){
		lang = "pois.db";
	}

	generadorRuta Ruta = new generadorRuta(this, ruta);
	generadorRuta Pois = new generadorRuta(this, null, lang);

	try {
		this.AP.setPrimeraVez("no");
		Ruta.createDataBase();
		Pois.createDataBasePois();
	} catch (Exception e) {
		Ruta.close();
		Pois.close();
	}

	Ruta.close();
	Pois.close();

	Button cerrar = (Button) findViewById(R.id.cerrar);

	cerrar.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {
			cerrar(v);
		}
	});
}
```

Es en este punto en el que descodificamos las preferencias elegidas por el usuario, para poder crear las bases de datos correspondientes. Como ya se ha explicado anteriormente la creación de las mismas, vamos a pasar a lo siguiente. El usuario pulsa el botón que se le muestra en pantalla, se ejecuta la función *cerrar(vista)* y se lanza la última actividad del proceso, la de la clase *CSVersion2*, a la que hemos pasado las preferencias del usuario de esta forma:

```
Intent i = new Intent(this, CSversion2.class);
i.putExtra("ruta", ruta);
i.putExtra("updates", upd);
i.putExtra("language", lang);
startActivity(i);
```

Con este código conseguimos enviar información de una actividad a otra. Se pueden enviar objetos, aunque en este caso con enviar ciertas palabras puede sernos suficiente, ya que no requerimos de gran cantidad de información. Con todo ello, finalmente conseguimos lanzar la actividad que va a procesar todos los datos y que ordenará la creación de las rutas y los puntos de interés, y gestionará el funcionamiento de la flecha.

Hasta ahora el funcionamiento de la aplicación se realizaba en sentido vertical, es decir, con el dispositivo de pie, pero para aprovechar mejor el tamaño de la pantalla y obtener una mayor visión, obligamos al dispositivo a que ejecute la aplicación de forma apaisada:

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

Posteriormente, se crean las rutas y los puntos de interés como ya se ha comentado anteriormente, y a continuación se dispone a preparar la superficie OpenGL ES donde mostraremos la flecha. Para ello, primero se crea la superficie y se le asigna un renderizador que será el que dibuje los elementos en dicha superficie.

```
// superficie GL para mostrar la flecha 3D
mGLSurfaceView = new GLSurfaceView(this);
mGLSurfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
CompassRenderer compassRenderer = new CompassRenderer(this);
mGLSurfaceView.setRenderer(compassRenderer);
mGLSurfaceView.getHolder().setFormat(PixelFormat.TRANSLUCENT);
this.setContentView(mGLSurfaceView);
```

Con la última línea de código hacemos constar al sistema que queremos que la vista o superficie OpenGL ES que hemos creado sea la que se muestre en el dispositivo. Con esto ya conseguimos que se dibuje la flecha en la pantalla.

Lo siguiente es añadir los elementos de la realidad aumentada a la vista del dispositivo, que son el radar, donde se muestra la localización de los puntos de interés en función de nuestra dirección, y la barra vertical que permite al usuario indicar la distancia a la que se muestran puntos de interés.

Algo que hemos comentado anteriormente que realizaba otra clase, era aplicar un filtro de paso bajo para suavizar el movimiento de la flecha y que no se moviera excesivamente con sólo movernos un poco. Por ello, debemos registrar los sensores que intervienen en este proceso, que son el acelerómetro y el sensor magnético. A pesar de que se deben registrar dichos sensores en la clase *CSVersion2*, es en la clase *CompassRenderer* en la que se tiene que administrar sus datos. La funcionalidad de esta clase está centrada en la renderización de la flecha, cosa que ya hemos comentado previamente.

LECTOR DE ARCHIVOS OBJ

La clase *CompassRenderer* hace uso de otra clase llamada *Arrow* para renderizar la flecha. En dicha clase se realiza un parseo de un archivo .obj, el cual contiene los diferentes parámetros que definen un objeto 3D. A continuación vamos a ver como está estructurado dicho archivo y como hemos realizado el proceso de lectura para generar la flecha.

```
# Blender v2.62 (sub 0) OBJ File: ''
# www.blender.org
mtllib flecha.mtl
o Pyramid01
v 0.072177 -55.401306 -0.037022
v -11.839181 -33.794655 -8.624275
v 4.513446 -0.436035 -3.624810
v 4.513446 -33.794659 -3.238863
vt 0.316072 0.316072
vt 0.857030 0.000000
vt 0.857030 0.092870
vt 0.857030 1.000000
vn -0.000000 -0.369335 -0.929296
vn 0.000000 0.000000 -1.000000
vn -1.000000 0.000000 -0.000000
vn 1.000000 0.000000 0.000000
usemtl
s off
f 1/1/1 2/2/1 3/3/1
f 1/1/2 18/4/2 4/5/2
f 1/1/3 19/2/3 5/3/3
```

Este fragmento es una pequeña muestra de cómo es un archivo .obj. Como ya hemos comentado se había realizado la flecha en 3DS Max, y posteriormente exportado a Blender, y es por ello que aparece el nombre del segundo programa en el archivo. Vamos a explicar que quiere decir cada una de las siglas que aparecen en el archivo:

- **v:** son los vértices que forman la figura 3D que hemos realizado. Cada uno de los valores que aparecen a continuación son las coordenadas en las que se sitúa cada vértice. En este caso el primer vértice estaría en la posición (`0.072177, -55.401306, 0.037022`) del plano.

- **vt:** son las coordenadas de las texturas de la figura. En este caso para nuestra aplicación no es necesario hacer uso de ellas.

- **vn:** son las normales de los vértices. Tampoco las necesitamos en nuestra aplicación, aunque en conjunto con las anteriores podríamos mejorar la apariencia del objeto con sombras e iluminación.

- **f:** son las caras que forman el objeto 3D. Para entender como se describen, veamos la segunda línea, "`f 1/1/2 18/4/2 4/5/2`". Ello quiere decir que la cara la componen tres vértices, el primero con la primera textura y la segunda normal, el decimoctavo vértice con la cuarta textura y la segunda normal y el cuarto vértice con la quinta textura y la segunda normal.

CREACIÓN DE BD A PARTIR DE ARCHIVOS GPS

En anteriores apartados ya hemos comentado como era el proceso de creación de las rutas y los puntos de interés a partir de las bases de datos de los mismos. Todas las bases de datos, tanto de las rutas como de los puntos de interés, no se han encontrado directamente como tal, si no que para obtenerlas se ha seguido un proceso de búsqueda de la ruta a través de Internet, consiguiendo en algunos casos rutas reducidas a partir de las que se podrían encontrar para dispositivos GPS.

Pero todo lo encontrado no era ya la base de datos como tal, sino que había que generarlas a partir de archivos gpx. Estos archivos, una vez abiertos tienen la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx
  version="1.0"
  creator="GPSBabel - http://www.gpsbabel.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.topografix.com/GPX/1/0"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd">
<time>2012-07-23T09:01:58Z</time>
<bounds minlat="42.880619000" minlon="-8.544348000" maxlat="43.572889000"
maxlon="-1.784438000"/>
<trk>
  <name>Camino de Santiago por la Costa o Camino Norte </name>
  <desc>Camino de Santiago por la Costa o Camino Norte completo, desde Irún a
Santiago de Compostela. </desc>
<trkseg>
<trkpt lat="43.349612000" lon="-1.784438000">
  <ele>14.000000</ele>
  <time>2007-06-22T08:21:15Z</time>
</trkpt>
<trkpt lat="43.338270000" lon="-1.788296000">
  <ele>5.000000</ele>
  <time>2007-06-22T08:23:21Z</time>
</trkpt>
<trkpt lat="43.351601000" lon="-1.812572000">
  <ele>-1.000000</ele>
  <time>2007-06-22T08:29:26Z</time>
</trkpt>
<trkpt lat="43.353554000" lon="-1.806993000">
  <ele>18.000000</ele>
  <time>2007-06-22T08:34:33Z</time>
</trkpt>
```

Este fragmento de código es un ejemplo de cómo son los archivos que hemos encontrado para obtener las rutas, en este caso se corresponde a la variante del Camino Norte del Camino de Santiago.

Para extraer cada punto de la ruta y generar así una base de datos que los contenga, ha sido necesario realizar otro programa externo a la aplicación que realizara esta función, para liberar de carga de memoria.

Este nuevo programa consiste en obtener un archivo de la carpeta de recursos del proyecto, y parsear el contenido del fichero como si fuera un archivo XML. Para ello se han seguido los siguientes pasos:

Se obtiene el archivo de la carpeta “res/raw” del proyecto y se parsea con ayuda de las clases que nos ofrece el API de Android.

```
Resources res = context.getResources();
InputStream fraw =
res.openRawResource(R.raw.caminosantiagonorteyprimitivo_gpx);
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document d = db.parse(fraw);
d.getDocumentElement().normalize();
```

Ahora tenemos que conseguir una lista con los nodos del fichero a partir de la etiqueta que nos interese, en este caso “`trkpt`”.

```
NodeList listaRutas = d.getElementsByTagName("trkpt");
int numRutas = listaRutas.getLength();
```

A partir de ahora ya podemos acceder a cada uno de los nodos del fichero individualmente, para extraer la información del punto en cuestión e ir formando así la base de datos que guarda todos los puntos de la ruta.

```
for(int i = 0; i < numRutas; i++){
       n = listaRutas.item(i);

       NamedNodeMap atr = n.getAttributes();
       Node nodelat = atr.getNamedItem("lat");
       Node nodelong = atr.getNamedItem("lon");

       String lat = nodelat.getNodeValue();
       String longi = nodelong.getNodeValue();

       ContentValues ins = new ContentValues();
       ins.put(LATITUD, lat);
       ins.put(LONGITUD, longi);

       database.insertOrThrow(TABLE_NAME, null, ins);
}
```

De cada nodo cogemos los atributos de la latitud y la longitud, y mediante consultas SQL vamos generando la base de datos, la cual se almacena en la partición “data” igual que las preferencias de la aplicación. Gracias a Eclipse [6] podemos acceder fácilmente a dicha partición y extraer la base de datos una vez generada.

Ahora sólo tenemos que incluirla en el proyecto de nuestra aplicación y tratarlas como hemos comentado anteriormente para incluirlas en la aplicación.

6 – PRUEBAS DE CONCEPTO

6.1 Realidad aumentada

En apartados anteriores ya se ha comentado como se decidió presentar los elementos de realidad aumentada de la aplicación, con Canvas para los puntos de interés y OpenGL ES para la flecha. Pero así como para la renderización de la flecha se decidió rápidamente optar por OpenGL ES, para los puntos de interés se intentaron diversas formas de mostrarlos en forma de realidad aumentada.

Una de las aplicaciones referentes en este campo al comienzo de desarrollo del proyecto era Layar. Layar es una aplicación cuya finalidad es permitir al usuario ver ciertas localizaciones en realidad aumentada.

Ilustración 25 – Puntos de interés en Layar

Layar funciona de la siguiente manera: se crea una capa con información sobre los puntos de quiere mostrar y el usuario que quiera hacer uso de ella debe hacerlo a través de su aplicación, por lo que las opciones de adaptabilidad son escasas y no nos permite incluir su tecnología en nuestra aplicación, motivo por el cual se descartó usar Layar.

Como ya vimos en la introducción, una de las aplicaciones que nos podía servir como punto de referencia era Wikitude, un GPS para Android en realidad aumentada. La parte de dicha aplicación orientada a mostrar localizaciones en vez de ruta tiene la forma siguiente:

Ilustración 26 - Puntos de interés en Wikitude

Como vemos en la imagen, dispone de un radar, un icono que señala la localización de un punto de interés y una caja de texto donde se muestra la información de dicho punto. Todo ello junto era aproximadamente la información que se había pensado para mostrar los puntos de interés de nuestra aplicación, por lo que se investigó como se podía emplear su software para nuestro objetivo.

Wikitude es una aplicación GPS parecida a los convencionales, pero nos ofrece un SDK que nos permite añadir realidad aumentada a nuestras aplicaciones Android. Parecía una buena forma de hacerlo y muy poco intrusiva por lo que se decidió hacer uso de él.

El código necesario para incluir los puntos de interés de realidad aumentada en nuestra aplicación es muy sencillo, empleando sus librerías y con sólo incluir las siguientes líneas de código ya disponemos de ellos en nuestra aplicación:

```
private void addPois(WikitudeARIntent intent) {
	WikitudePOI poi1 = new WikitudePOI(35.683333, 139.766667, 36, "Tokyo",
"Tokyo is the capital of Japan.");
	poi1.setLink("http://www.tourism.metro.tokyo.jp/");
	poi1.setDetailAction(BasicOpenARDemoActivity.CALLBACK_INTENT);

	List<WikitudePOI> pois = new ArrayList<WikitudePOI>();

	pois.add(poi1);
	intent.addPOIs(pois);

	((BasicOpenARDemoApplication) this.getApplication()).setPois(pois);
}
```

Básicamente realizando una llamada a esta función ya dispondríamos de los puntos que deseáramos en nuestra aplicación y de forma muy simple.

Para poder hacer uso del SDK de Wikitude debemos registrarnos como desarrolladores en su Web y a cambio nos dan una clave que tenemos que incluir en nuestra aplicación.

Esto es gratuito siempre y cuando nuestra aplicación no vaya a ser de pago o sea de prueba, en caso contrario tendríamos que pagarles para poder usar su SDK. A priori esto no supone ningún contratiempo, ya que nuestra aplicación es gratuita. Desde Wikitude, para que una aplicación pueda usar su SDK, requieren de una comprobación de la clave cada vez que se hace uso de dicha aplicación.

Esta comprobación se tiene que realizar a través de Internet mediante una conexión a sus servidores, lo que supone un problema ya que nuestra aplicación está pensada para ser utilizada sin necesidad de tener una tarifa de datos y que al usuario le suponga dinero. En caso de no poder realizar la comprobación de la clave nos aparece un mensaje muy molesto que ocupa casi toda la pantalla y que impide la correcta utilización de la aplicación, motivo por el cual se descartó emplear Wikitude para realizar esta función.

Los problemas que surgieron hicieron que se buscará un SDK o Framework open source y que los evitaran. Tras cierto tiempo buscando y probando diferentes Frameworks, se optó por emplear Mixare. Mixare es un navegador en realidad aumentada open source gratuito que nos ofrece toda la funcionalidad que estábamos buscando.

Existen múltiples formas de usar Mixare en conjunto con nuestra aplicación, incluso a través de páginas Web, pero también ofrece la posibilidad de usar nuestras propias bases de datos.

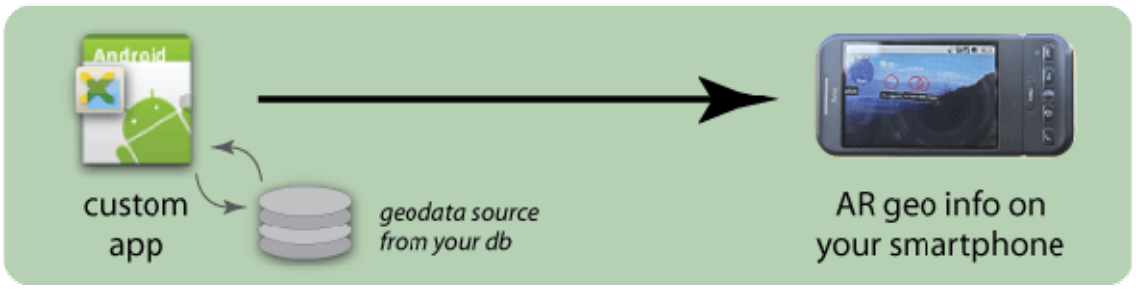

Ilustración 27 - Modo de uso de la librería Mixare

Finalmente se tomó el Framework de Mixare para realizar nuestra aplicación, y el resultado final ya se ha mostrado anteriormente.

6.2 Realizando una ruta del Camino de Santiago

La funcionalidad principal y más importante de la aplicación es guiar al usuario a través del Camino de Santiago, pero para llegar a cumplir dicho objetivo es necesario definir las pautas que se van a seguir para considerar una ruta como realizada.

En un principio se pensó en la aplicación como un GPS más, que estuviera conectado mientras se realizara la ruta, por lo que las rutas que se buscaron fueron las que los GPS de hoy en día utilizan. Las rutas seleccionadas constaban de aproximadamente 20.000 puntos GPS. Tantos puntos son más que suficientes para señalar una ruta y lo obvio es empezar por el principio, por el primer punto de la misma.

Dado que las rutas se habían guardado en bases de datos, lo más sencillo era realizar una consulta SQL e ir cogiendo los puntos uno a uno para formar la ruta. Para ver si el usuario ha alcanzado uno de los puntos y apuntar así al siguiente existe una función de la clase *Location* que nos indica la distancia a la que se encuentra la localización en función de nuestra localización.

Si el punto al que nos estamos dirigiendo se encuentra a menos de 20 metros por ejemplo, se considera que ya se ha llegado (debido a la posible imprecisión de la señal GPS), se realiza una nueva consulta en la base de datos y se obtienen la latitud y la longitud del siguiente punto al que hemos llegado.

Esto funcionaría correctamente en caso de realizar toda la ruta sin parar, por lo que hay que considerar que el usuario deje de utilizar la aplicación o que apague el teléfono, teniendo que guardar la última posición a la que hemos llegado. Para ello, a parte de la primera vez que se inicia la aplicación, hay que ir actualizando la última posición a la que se llega por si se detiene la ejecución. Se decide emplear un fichero en el que guardar dicha posición y almacenarlo en la memoria externa del dispositivo.

```
File ruta_sd = Environment.getExternalStorageDirectory();
File f = new File(ruta_sd.getAbsolutePath(), "posicion_brujula.txt");
OutputStreamWriter fout = new OutputStreamWriter(new FileOutputStream(f));
fout.write(this.posicion);
fout.close();
```

Con ello accedemos a la tarjeta de memoria del dispositivo y actualizamos el valor de la posición en el fichero “`posicion_brujula.txt`”, asegurándonos de siempre que tenemos la última posición guardada. Si el usuario ha dejado de usar la aplicación no tenemos más que leer de dicho archivo, tomar la posición y hacer una consulta en la base de datos de la ruta para saber cual es el siguiente punto a indicar.

Con este sistema parece resuelta la incógnita de cómo ir señalando el Camino, pero existen ciertos problemas que pueden resultar negativos para el funcionamiento de la aplicación. Para empezar este sistema hace fundamental que el usuario esté continuamente utilizando la aplicación, ya que necesitamos llegar a un punto de la ruta para considerar que lo hemos alcanzado y apuntar de esa forma a la siguiente localización.

Luego ocurre que como disponemos de más de 19.000 puntos GPS, todos están muy juntos unos con otros, y necesitamos actualizaciones muy seguidas de la señal GPS para considerar que hemos pasado de uno a otro. Estos dos problemas hacen que algo tan crítico como la batería de un dispositivo de estas características se acabe enseguida.

Ante esta circunstancia podrían ocurrir dos cosas: el usuario deja de utilizar la aplicación o el dispositivo se queda sin batería. La consecuencia directa de ambas posibilidades es que la aplicación no sabría si hemos llegado a un punto de la ruta o no, por lo que por ejemplo si empezamos a hacer el Camino de Santiago sin utilizar la aplicación, y varios días después la encendemos para ver hacia dónde tenemos que ir, nos señalaría al primer punto de toda la ruta y no verdaderamente al siguiente punto al que dirigirse.

Ello hizo replantearse el funcionamiento completo de la aplicación, optando finalmente por el sistema que actualmente se emplea, guardar todos los puntos de la ruta en una lista de marcadores, ordenarlos según la distancia a la que nos encontremos de ellos y hacer que la flecha indique el camino más corto al punto más cercano de la ruta.

Esto permite que el usuario pueda no utilizar la aplicación constantemente, si no que lo haga cuando verdaderamente necesite encontrar el camino porque se ha perdido, o porque ha llegado a una ciudad y necesita ver donde se encuentra el hostal más cercano.

Además, gracias al sitio Web de Wikiloc **[** 4**]**, una página en la que cualquier persona puede subir las rutas que ha empleado para que otros las realicen, se encontraron diversas rutas del Camino de Santiago con la posibilidad de reducir el número de punto de las mismas, pasando de tener rutas de más de 19.000 puntos hasta conseguir disponer de únicamente 500 puntos, aligerando los tamaños de las bases de datos, reduciendo el número de consultas a realizar significativamente y sobre todo, permitiendo al dispositivo reducir el gasto de energía, puesto que el período en que se necesitaban recibir las actualizaciones pasaba de casi cada 5 o 10 segundos, a permitir al usuario elegir un período máximo de 5 minutos.

7 – Conclusiones

El objetivo principal de este proyecto era realizar una aplicación novedosa para dispositivos móviles, que fuera útil para los usuarios y no se convirtiera en una aplicación un solo uso únicamente, si no que se les permitiera utilizarla a lo largo del tiempo.

Creo sinceramente que el objetivo se ha visto cumplido, ya que se ha realizado una aplicación para Android, uno de los sistemas operativos para smartphones más empleados en el mundo, con el fin de ayudar a sus usuarios a realizar algo tan personal y multicultural como es el Camino de Santiago, lo que hace que no sea una aplicación centrada en un público concreto, si no que cualquier persona que desee realizarlo puede hacerlo sabiendo que tiene un apoyo tanto para no perderse como para encontrar un sitio en el que pasar la noche.

Además se aleja de lo convencional en cuanto a GPS se refiere, puesto que no es simplemente algo digital, si no que plasma en lo que realmente estamos viendo un camino lleno de lugares de interés que forman parte del Camino de Santiago haciendo uso de la realidad aumentada.

Y dado que el código del proyecto está completado, ya se ha puesto a disposición de los usuarios de Android en la Play Store, contando con las siguientes estadísticas por ahora.

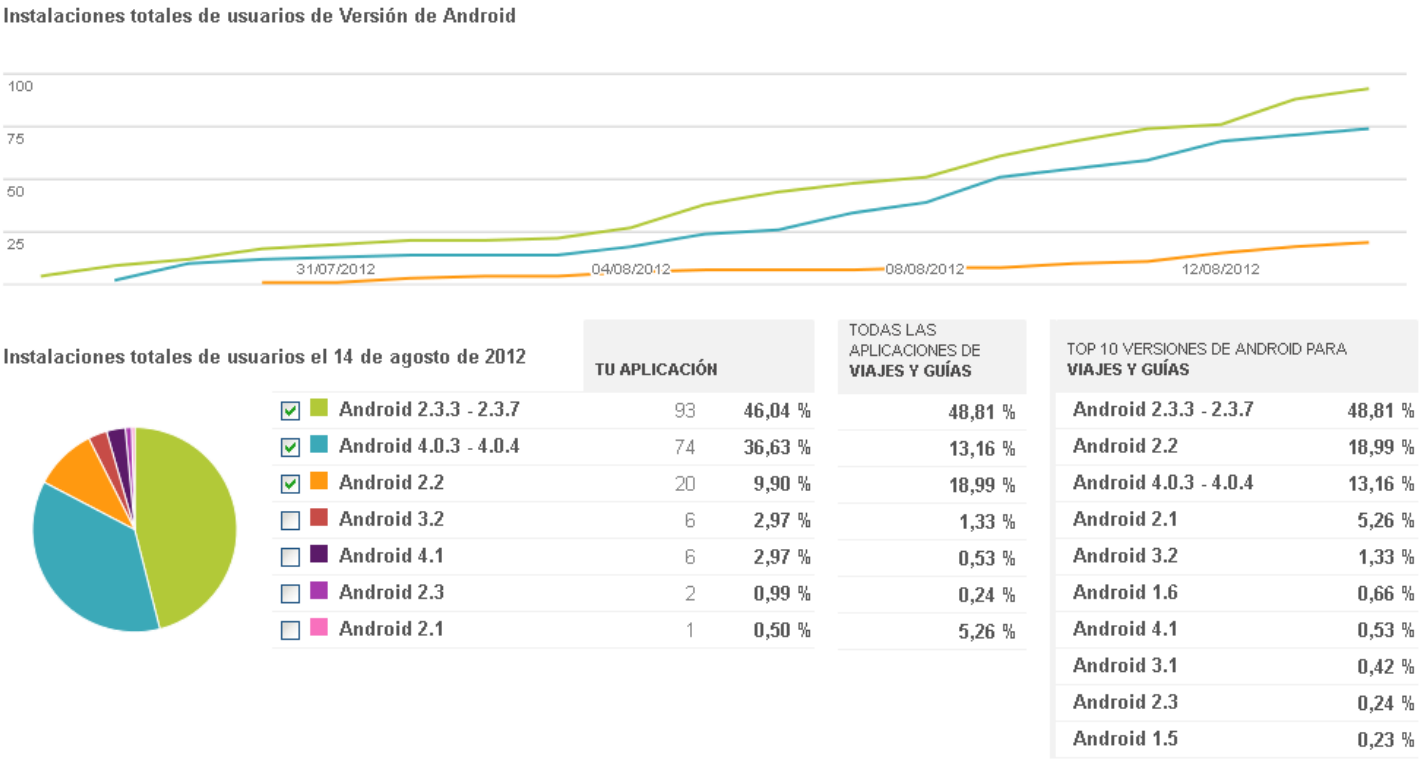

Ilustración 28 - Gráfico de descargas de la aplicación

Actualmente cuenta con algo más de 200 descargas, cosa con la que personalmente estoy muy contento ya que es una aplicación gratuita, destinada a que usuarios de cualquier edad la utilicen y con el único objetivo de ser de utilidad a todos los que realizan el Camino de Santiago.

Este proyecto ha resultado muy satisfactorio a nivel personal, puesto que me ha servido para enriquecer mis conocimientos técnicos, en algo tan en auge como son las plataformas móviles, permitiéndome adquirir experiencia que de otra forma no conseguiría, ya que a lo largo de estos años en la Universidad te enseñan a realizar programas, pero no aplicaciones destinadas al público en general.

También me ha ayudado a saber administrar mejor mis recursos y mi tiempo, dado que se ha realizado al mismo tiempo que el último curso de Ingeniería Informática.

8 – Líneas futuras

Al ser el objetivo de este proyecto una aplicación de cara al usuario, todo lo que sean mejoras en la usabilidad o en la interfaz servirán como líneas futuras de la aplicación.

Por ejemplo, se podrían realizar mejoras en el diseño de la flecha 3D que indica el camino, ya que como se comprobaba en las imágenes de la misma al inicio del documento, el cambio en la apariencia de la flecha en el diseño y en la renderización es completamente distinta. Para lograr esto sería necesario investigar a fondo la tecnología de OpenGL ES, haciendo hincapié en lo referente a la iluminación, puesto que este es el motivo de no ver sombras en la flecha, que no empleamos las luces que nos ofrece OpenGL ES.

Siguiendo el hilo de las mejoras gráficas, se podría ofrecer al usuario la posibilidad de modificar a su gusto la apariencia de la aplicación. Una posibilidad sería ofertar diferentes diseños de flechas 3D para que el usuario escoja la que más le guste, así como los iconos de cada punto de interés, etc.

Dado que la aplicación pretende guiar al usuario a través del Camino de Santiago nos podríamos centrar en las rutas del mismo. Actualmente la aplicación cuenta con tres rutas: ruta francesa, ruta del norte/costa y ruta del norte con variante por el camino primitivo. Existen multitud de variantes que no se han recogido por lo que hasta completar el número de las existentes podríamos mejorar dicho aspecto. La aplicación cuenta además con más de 200 puntos de interés repartidos a lo largo de todo el camino, pero se podrían incluir muchos más.

Si nos fijamos en la imagen del apartado anterior, vemos en el gráfico las diferentes versiones de Android en las que se encuentra instalada la aplicación, Este proyecto se ha realizado para la versión 2.2 de Android, a pesar de que las pruebas también se han realizado en la versión 2.3.3, por lo que no deberían producirse problemas en esta versión. Pero el resto de versiones, sobre todo las más nuevas a partir de la 4.0, es posible que se produzcan ciertos fallos que no se han previsto, por lo que sería recomendable realizar diversas pruebas con dichas versiones del sistema operativo.

Por último, dada la multiculturalidad del Camino de Santiago, y que es una actividad que relaciona mucho a las personas, se podría considerar añadir ciertos temas de interacción entre los usuarios de la aplicación, como poder añadir amigos y saber donde están, recomendar cierto punto de interés en el que hemos estado, recomendar a un amigo la aplicación, etc., haciendo mucho más interesante el uso de la aplicación y diversificando su actividad.

9 – Bibliografía

[1] Purdy, K., (2010), The Complete Android Guide.

[2] Gargenta, M., (2011), Learning Android, O'Reilly, California.

[3] Stack Overflow, <http://www.stackoverflow.com>

[4] Wikiloc – Rutas y puntos de interés GPS del Mundo, <http://es.wikiloc.com>

[5] Wright, R., Lipchak, B., (2005), OpenGL (Programación), Anaya Multimedia, Madrid.

[6] Eclipse - The Eclipse Foundation open source community website, <http://www.eclipse.org/>

[7] Android SDK | Android Developers, <http://developer.android.com/sdk/index.html>